SyRep: Efficient Synthesis and Repair of Fast Re-Route Forwarding Tables for Resilient Networks

Csaba Györgyi, Kim G. Larsen, Stefan Schmid and Jiri Srba

Aalborg University, Denmark
Technical University Berlin, Germany
University of Vienna, Austria

DSN 2024, Brisbane, Australia

Logos

universität
wien

Motivation

- Nowadays communication networks must be highly resilient.
- In modern communication networks with stringent dependability requirements, local fast re-routing (FRR) is essential for a quick response to link failures.
- Configuring FRR for multiple failures is, however, challenging since a router's forwarding table may take into account only the failed links directly incident to it.
- We focus on **repairing** existing routing configurations.

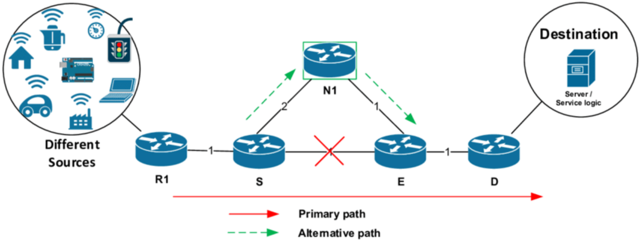


Figure by Papan et al., Sensors'2020.

Motivation

- Nowadays communication networks must be highly resilient.
- In modern communication networks with stringent dependability requirements, local fast re-routing (FRR) is essential for a quick response to link failures.
- Configuring FRR for multiple failures is, however, challenging since a router's forwarding table may take into account only the failed links directly incident to it.
- We focus on **repairing** existing routing configurations using BDDs because:
  - Networks in practice already come with certain mechanisms and forwarding tables
  - This method can be used to efficiently *synthesize* forwarding tables *from scratch*, using a hybrid approach: first, using a fast heuristic that provides close-to-resilient routing tables, and then quickly repair the ill-defined entries using our rigorous BDD approach.

How to react to link failures

Link failures are common in ISP networks as well as data centers.

- A failed link is detected first by the connected routers.
- The routers then usually initiate a reconfiguration of the network.
- Before the process converges, local fast failover protection kicks in.

Local Fast Re-Route (FRR)

If a router is supposed to forward a packet on an interface that is down, it uses instead a preinstalled backup forwarding rule.

Different technologies support FRR.

- **IP fast reroute** uses an alternative ECMP (Equal Cost Multi-Path), loop-free alternative path or multi-hop repair paths.
- In **MPLS networks**, a backup label is pushed on the label stack that goes around the failed link/node (link or facility protection).
- **OpenFlow** uses conditional failover rules with a prioritized list of next-hop interfaces.

Our FRR Model and Perfect Resilience

We study a basic fast re-route protection method where

- packet forwarding is driven by its destination only (no source matching),
- packet headers are not modified, and
- routers store "skipping" forwarding tables: prioritized list of next-hop interfaces per each incoming link and each packet destination.

Perfect Resilience

Our aim is to repair a given network topology's routing configuration (or synthesize it from scratch using a hybrid approach) so that in any failure scenario a packet is delivered to the destination as long as the connectivity between the source and destination is preserved.

Perfect Resilience

Perfectly k-Resilient Routing

A routing table is perfectly k-resilient for a given destination d if

- for any set F of failed links where $|F| \leq k$,
- a packet arriving to any node v is delivered to d as long as v and d are physically connected under the failure scenario F.

A routing is perfectly resilient if this holds for all k.

There are network topologies are not perfectly resilient.

- Feigenbaum et al. [PODC'12] show a network with 12 nodes, 22 edges that is not perfectly 14-resilient.
- Foerster et al. [APOCS'21] show a network with 6 nodes and 9 edges (K3,3) that is not perfectly 3-resilient.
- We prove that every network is perfectly 1-resilient (even if the primary path is an arbitrary chosen shortest path).
- It is an open question whether every network is perfectly 2-resilient.

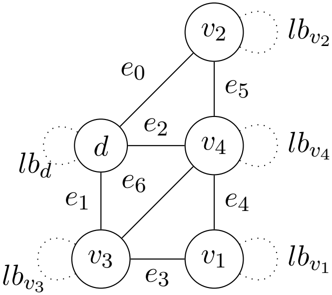
Main Contributions

We design and implement *SyRep*, an efficient and automated approach to repair and synthesize highly dependable fast re-routing tables, providing perfect k-resilience. This is an extension of SyPer that was published on INFOCOM'24.

- We define and implement an automatic method for repairing non-resilient routing tables using BDDs.
- We suggest a fast heuristic algorithm for populating skipping routing tables as well as a method for reducing the size of the synthesized networks by applying structural reduction rules.
- We implement the proposed methodology in a prototype tool SyRep and carry on a comparative experimental evaluation on a large range of the network topologies from the Internet Topology Zoo

Repairing with an Example

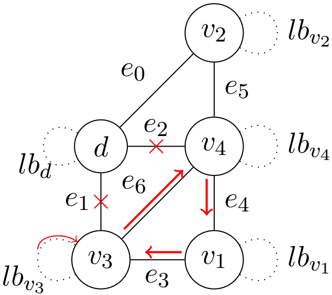
This is a topology and a corresponding forwarding table using skipping routing resilient up to one failure.



in-edge	node	out-edges
lb_{v_1}	v_1	e_3,e_4
e_3	v_1	e_4, e_3
e_4	v_1	e_3,e_4
lb_{v_2}	v_2	e_0, e_5
e_5	v_2	e_0, e_5
lb_{v_3}	v_3	e_1,e_6,e_3
e_3	v_3	e_1,e_6,e_3
e_6	v_3	e_1, e_3, e_6
lb_{v_4}	v_4	e_2,e_4,e_5
e_4	v_4	e_2, e_5, e_6
e_5	v_4	e_2, e_4, e_6
e_6	v_4	e_2,e_4,e_5

Repairing with an Example

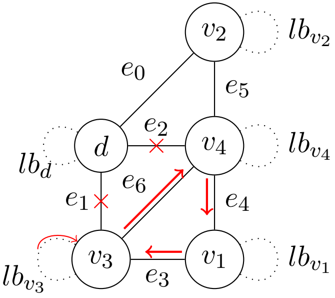
However, this configuration is not resilient to every two link failures. Our method investigates all possible scenarios and marks the used entries.



in-edge	node	out-edges
lb_{v_1}	v_1	e_3, e_4
e_3	v_1	e_4, e_3
e_4	v_1	e_3, e_4
lb_{v_2}	v_2	e_0, e_5
e_5	v_2	e_0, e_5
lb_{v_3}	v_3	e_1, e_6, e_3
e_3	v_3	e_1, e_6, e_3
e_6	v_3	e_1, e_3, e_6
lb_{v_4}	v_4	e_2, e_4, e_5
e_4	v_4	e_2, e_5, e_6
e_5	v_4	e_2, e_4, e_6
e_6	v_4	e_2, e_4, e_5

Repairing with an Example

Using BDDs we try to replace the marked entries with new ones. Indeed, if we erase e_4 from the last row (i.e. replaced with e_5), the configuration becomes perfectly 2-resilient.

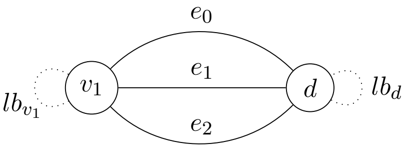


in-edge	node	out-edges
lb_{v_1}	v_1	e_3, e_4
e_3	v_1	e_4, e_3
e_4	v_1	e_3, e_4
lb_{v_2}	v_2	e_0, e_5
e_5	v_2	e_0, e_5
lb_{v_3}	v_3	e_1, e_6, e_3
e_3	v_3	e_1, e_6, e_3
e_6	v_3	e_1, e_3, e_6
lb_{v_4}	v_4	e_2, e_4, e_5
e_4	v_4	e_2, e_5, e_6
e_5	v_4	e_2, e_4, e_6
e_6	v_4	$e_2, \cancel{e_4}, e_5$

How does a BDD look like? (an even smaller example)

Idea

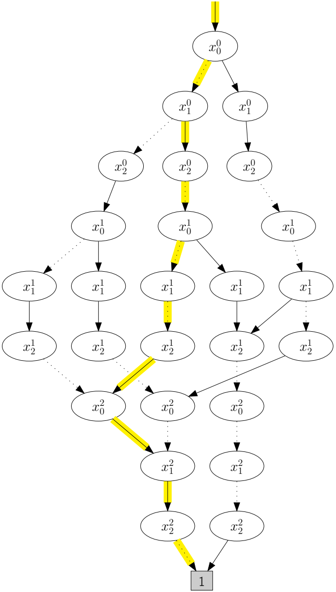
Represent routing tables as Boolean functions and store them in the compact data structure Binary Decision Diagrams (BDD).



Edges are ordered as: lb_d, lb_{v_1}, e_0 , e_1 , e_2

Highlighted path encodes the priority list:

$$x_2^0 x_1^0 x_0^0 = 010 \rightarrow e_0$$
$$x_2^1 x_1^1 x_0^1 = 100 \rightarrow e_2$$
$$x_2^2 x_1^2 x_0^2 = 011 \rightarrow e_1$$



BDD formulation

Encoding of next-hop

$$\mathcal{T}(\overline{\mathbf{e}}_{in}, \overline{\mathbf{v}}, \overline{\mathbf{e}}_{out}, \overline{\mathbf{v}}', \overline{\mathbf{f}}_1, ..., \overline{\mathbf{f}}_k, [\overline{\mathbf{e}}^0_{v,e}, \overline{\mathbf{e}}^1_{v,e}, ..., \overline{\mathbf{e}}^k_{v,e} : {}^{e\in E}_{v\in r(e)}]) =$$

$$state(\overline{\mathbf{v}}, \overline{\mathbf{e}}_{in}) \wedge state(\overline{\mathbf{v}}, \overline{\mathbf{e}}_{out}) \wedge state(\overline{\mathbf{v}}', \overline{\mathbf{e}}_{out}) \wedge$$

$$\overline{\mathbf{e}}_{out} \notin \{\overline{\mathbf{f}}_1, ..., \overline{\mathbf{f}}_k\} \wedge$$

$$\bigwedge_{e\in E \wedge v\in r(e)} \mathcal{V}_{v,e}(\overline{\mathbf{e}}^0_{v,e}, \overline{\mathbf{e}}^1_{v,e}, ..., \overline{\mathbf{e}}^k_{v,e}) \wedge$$

$$\bigwedge_{e\in E} \bigwedge_{v\in r(e)} \Big(\overline{\mathbf{e}}_{in}(e) \wedge \overline{\mathbf{v}}(v) \implies \overline{\mathbf{e}}_{out} \in \{\overline{\mathbf{e}}^0_{v,e}, ..., \overline{\mathbf{e}}^k_{v,e}\}\Big) \wedge$$

$$\bigwedge_{i\in 0,..,k} \bigwedge_{e\in E} \bigwedge_{v\in r(e)} \Big(\overline{\mathbf{e}}_{in}(e) \wedge \overline{\mathbf{v}}(v) \wedge \overline{\mathbf{e}}^i_{v,e} \notin \{\overline{\mathbf{f}}_1, ..., \overline{\mathbf{f}}_k\} \wedge$$

$$\{\overline{\mathbf{e}}^0_{v,e}, ..., \overline{\mathbf{e}}^{i-1}_{v,e}\} \subseteq \{\overline{\mathbf{f}}_1, ..., \overline{\mathbf{f}}_k\} \implies \overline{\mathbf{e}}_{out} = \overline{\mathbf{e}}^i_{v,e}\Big)$$

Encoding of packet delivery

$$\mathcal{D}_{n+1}(\overline{\mathbf{e}}_{in}, \overline{\mathbf{v}}, \overline{\mathbf{f}}_1, ..., \overline{\mathbf{f}}_k, [\overline{\mathbf{e}}^0_{v,e}, \overline{\mathbf{e}}^1_{v,e}, ..., \overline{\mathbf{e}}^k_{v,e} : {}^{e\in E}_{v\in r(e)}]) =$$

$$\mathcal{D}_n(\overline{\mathbf{e}}_{in}, \overline{\mathbf{v}}, \overline{\mathbf{f}}_1, ..., \overline{\mathbf{f}}_k, [\overline{\mathbf{e}}^0_{v,e}, \overline{\mathbf{e}}^1_{v,e}, ..., \overline{\mathbf{e}}^k_{v,e} : {}^{e\in E}_{v\in r(e)}]) \vee$$

$$\Big(\exists \overline{\mathbf{v}}', \overline{\mathbf{e}}_{out} :$$

$$\mathcal{T}(\overline{\mathbf{e}}_{in}, \overline{\mathbf{v}}, \overline{\mathbf{e}}_{out}, \overline{\mathbf{v}}', \overline{\mathbf{f}}_1, ..., \overline{\mathbf{f}}_k, [\overline{\mathbf{e}}^0_{v,e}, \overline{\mathbf{e}}^1_{v,e}, ..., \overline{\mathbf{e}}^k_{v,e} : {}^{e\in E}_{v\in r(e)}])$$

$$\wedge \mathcal{D}_n(\overline{\mathbf{e}}_{out}, \overline{\mathbf{v}}', \overline{\mathbf{f}}_1, ..., \overline{\mathbf{f}}_k, [\overline{\mathbf{e}}^0_{v,e}, \overline{\mathbf{e}}^1_{v,e}, ..., \overline{\mathbf{e}}^k_{v,e} : {}^{e\in E}_{v\in r(e)}])\Big)$$

Encoding of all perfectly k-resilient routings

$$\mathcal{P}([\overline{\mathbf{e}}^0_{v,e}, \overline{\mathbf{e}}^1_{v,e}, ..., \overline{\mathbf{e}}^k_{v,e} : {}^{e\in E}_{v\in r(e)}]) =$$

$$\forall \overline{\mathbf{f}}_1, ..., \overline{\mathbf{f}}_k : \forall (\overline{\mathbf{e}}_s, \overline{\mathbf{s}}) :$$

$$\Gamma(s(\overline{\mathbf{s}}), f_1(\overline{\mathbf{f}}_1), ..., f_k(\overline{\mathbf{f}}_k), d) \implies$$

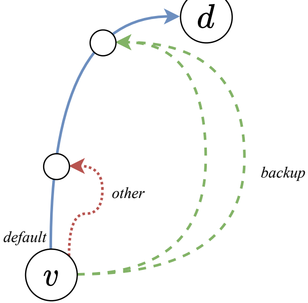
$$D(\overline{\mathbf{e}}_s, \overline{\mathbf{s}}, \overline{\mathbf{f}}_1, ..., \overline{\mathbf{f}}_k, [\overline{\mathbf{e}}^0_{v,e}, \overline{\mathbf{e}}^1_{v,e}, ..., \overline{\mathbf{e}}^k_{v,e} : {}^{e\in E}_{v\in r(e)}])$$

Fast generation of routing tables: Heuristic Routing Generation

Our heuristic generates close-to-resilient routings by extending the algorithm described in the SyPer paper. Formally, we define a skipping routing R such that for every edge $e \in E$ and every node $v \in V \setminus \{d\}$

- if $e \neq e_v$ then $R(e, v) := (e_v, e_1, e_2, \ldots, e_\ell, e'_1, e'_2, \ldots, e'_m, e)$
- otherwise $R(e_v, v) := (e_1, e_2, \ldots, e_\ell, e'_1, e'_2, \ldots, e'_m, e_v)$

where $e_1, e_2, \ldots, e_\ell$ are all backup edges for v (in an arbitrary order) and $e'_1, e'_2, \ldots, e'_m$ are all the remaining edges (different from e_v and the backup edges) connected to v.



Fast generation of routing tables: Structural Reduction Methods

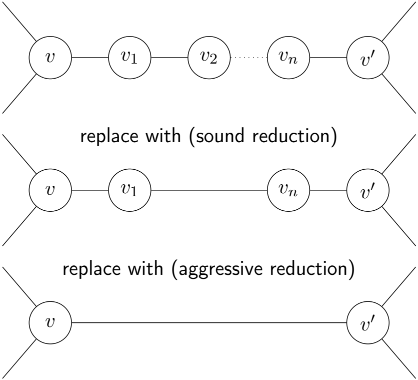
Reducing the size of the network can speed up the synthesis of resilient routing configurations. We present two reduction rules with different guarantees.

Sound chain-reduction:

- provably correct
- limited size reduction

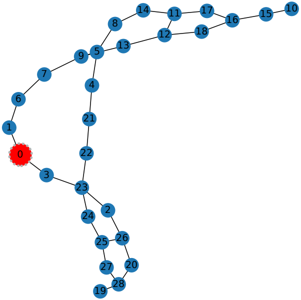
Aggressive chain-reduction:

- much more size reduction (super-set of sound version)
- the expanded version is not always resilient (but we can try to repair it)

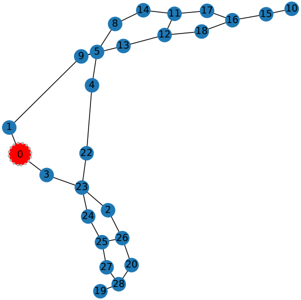


Structural Reduction Example

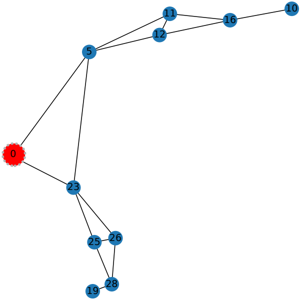
Effect of the two structural reduction rules on the *BizNet* topology where node 0 is the destination node.



Original (29 nodes)

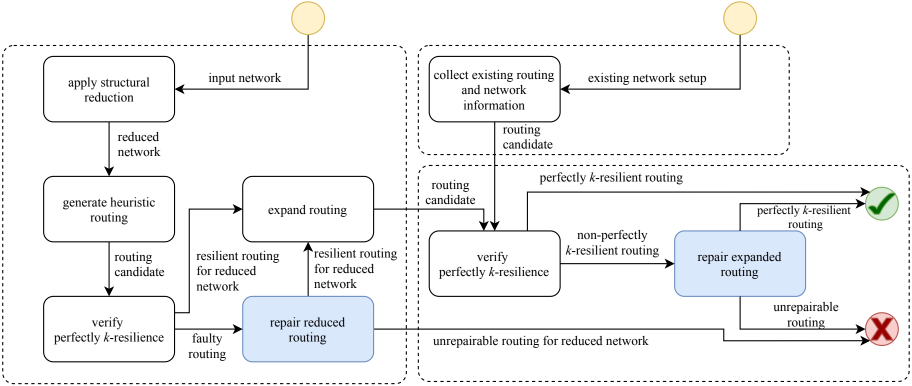


After sound reduction (26 nodes)



After aggressive reduction (11 nodes)

Combined framework with modular architecture

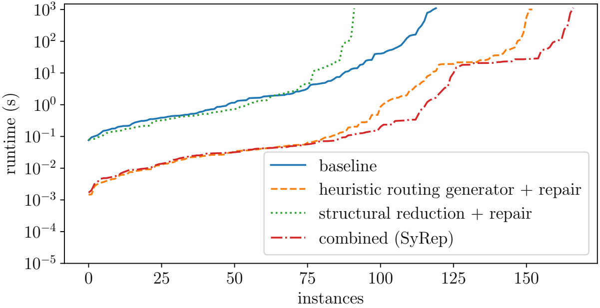


Implementation and experiments

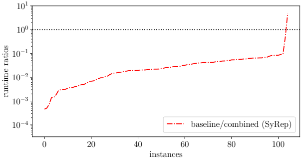
- SyRep implements in Python and uses the CUDD BDD library.
- Open-source tool, reproducibility package available.
- Evaluated on Topology Zoo benchmark of ISP network topologies (243 connected networks with up to 700 nodes).
- We try to find 2- and 3-resilient routings using BDDs with 20 minutes timeout and 128 GB memory limit.
- (Our baseline is the original SyPer tool.)

Topology Zoo benchmark $k = 2$

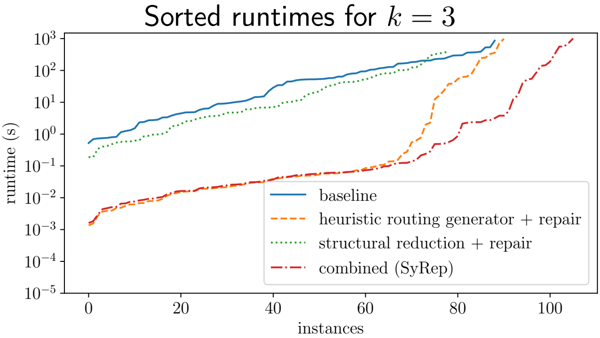
Sorted runtimes for $k = 2$

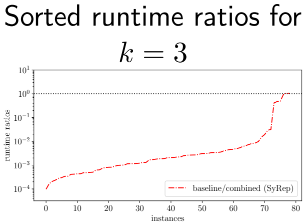


Sorted runtime ratios for $k = 2$

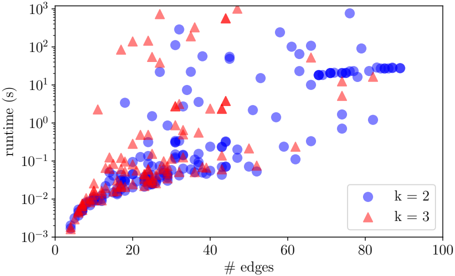


Topology Zoo benchmark $k = 3$

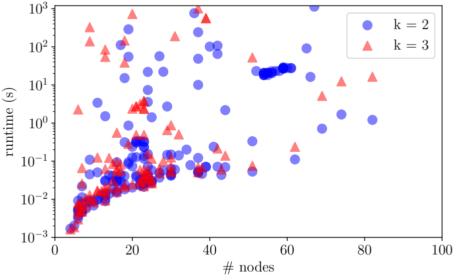




Network size vs. runtime



Network size (**number of edges**) vs. runtime



Network size (**number of nodes**) vs. runtime

Related Work: Resilient networking

- M. Chiesa, A. Kamisiński, J. Rak, G. Rétvári, and S. Schmid, "A survey of fast recovery mechanisms in the data plane," TechRxiv, May 2020.
- K.-T. Foerster, A. Kamisinski, Y.-A. Pignolet, S. Schmid, and G. Trédan, "Grafting arborescences for extra resilience of fast rerouting schemes," in INFOCOM 2021
  - this algorithm does not provide any formal guarantees
- The problem for graphs with certain properties is also widely studied.
- Fast re-routing solutions have also been studied for networks which support dynamic packet headers modifications and/or source matching.

Related Work: Synthesis and Formal methods

- Many interesting synthesis and formal methods approaches have been presented for other aspects of networking.
- *SyNET (CAV'17)* synthesizes correct network configurations for routing protocols such as BGP and OSPF
- *AalWines (CoNEXT'20)* provides an automated what-if verification of the policy-compliance of routes under multiple failures in MPLS networks.
- There are also tools to automatically and correctly update network configurations, such as *NetComplete (NSDI'18)* and *AllSynth (TASE'22)*, which ensure policy compliance.
  - AllSynth also leverages BDDs.

Conclusion and Open Questions

Contributions

- Quickly repairing existing forwarding tables using BDDs.
- Investigating heuristics to speed up the synthesis of routing configurations.
- Efficiently synthesize forwarding tables from scratch, using a hybrid approach.

Open questions:

- There exist other heuristic approaches for routing synthesis like e.g. Grafting. Are there heuristics with better/worse repairability?
- Can we use repairing to handle dynamically changing networks?
- As another future work, it will be interesting to extend our tool to provide additional desirable properties beyond connectivity, e.g., accounting for link utilization or congestion along backup paths.